

SAS/C® Development System Quick Reference Guide

Version 6.0



```
#include <graphics/gfxbase.h>
openlibrary("intuition.library")
library("graphics.library")
```


■ **SAS/C[®] Development System Quick Reference Guide**

Version 6.0



SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/C® Development System Quick Reference Guide, Version 6.0*, Cary, NC: SAS Institute Inc., 1992. 45 pp.

SAS/C® Development System Quick Reference Guide, Version 6.0

Copyright © 1992 by SAS Institute Inc., Cary, NC, USA.

ISBN 1-55544-498-9

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.
1st printing, July 1992

The SAS® System is an integrated system of software providing complete control over data access, management, analysis, and presentation. Base SAS software is the foundation of the SAS System. Products within the SAS System include SAS/ACCESS®, SAS/AF®, SAS/ASSIST®, SAS/CPE®, SAS/DMI®, SAS/ETS®, SAS/FSP®, SAS/GRAPH®, SAS/IML®, SAS/IMS-DL/I®, SAS/OR®, SAS/QC®, SAS/REPLAY-CICS®, SAS/SHARE®, SAS/STAT®, SAS/TOOLKIT®, SAS/CALC®, SAS/CONNECT®, SAS/DB2®, SAS/EIS®, SAS/ENGLISH®, SAS/INSIGHT®, SAS/LAB®, SAS/LOOKUP®, SAS/NVISION®, SAS/PH-Clinical®, SAS/SQL-DS®, and SAS/TUTOR™ software. Other SAS Institute products are SYSTEM 2000® Data Management Software, with basic SYSTEM 2000, CREATE®, Multi-User®, QueX®, Screen Writer®, and CICS interface software; NeoVisuals® software; JMP®, JMP IN®, and JMP Serve® software; SAS/RTERM® software; and the SAS/C® Compiler and the SAS/CX® Compiler. MultiVendor Architecture™ and MVA™ are trademarks of SAS Institute Inc. SAS Institute also offers SAS Consulting® and On-Site Ambassador™ services. SAS Communications®, SAS Training®, SAS Views®, the SASware Ballot®, and Observations™ are published by SAS Institute Inc. All trademarks above are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. The Institute is a private company devoted to the support and further development of its software and related services.

Amiga Includes and Development Tools Version 2.0

Copyright © 1990 Commodore-Amiga, Inc. All rights reserved.

Amiga Workbench™ Version 2.0

Copyright © 1988, 1990 Commodore-Amiga, Inc. All rights reserved.

Installer

Copyright © 1991-1992 Commodore-Amiga, Inc. All rights reserved.

AmigaGuide™ and WDisplay

Copyright © 1991-1992 Commodore-Amiga, Inc. All rights reserved.

Distributed under license from Commodore.

Amiga®, AmigaDOS®, Workbench®, and AmigaGuide™ are registered trademarks or trademarks of Commodore-Amiga, Inc. Commodore® is a registered trademark or trademark of Commodore Electronics Limited.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Doc S17, Ver112.6, 072292

Contents

v Using This Book

1 Chapter 1 Compiler Options

13 Chapter 2 Debugger Commands

21 Chapter 3 Library Functions

Using This Book

Purpose

SAS/C Development System Quick Reference Guide, Version 6.0 contains reference tables for the compiler and linker options, debugger commands, and library functions.

How This Book Is Organized

SAS/C Development System Quick Reference Guide has three chapters:

Chapter 1, “Compiler and Linker Options”

describes the options that you can specify on the `sc` and `slink` commands.

Chapter 2, “Debugger Commands”

lists the format of and options supported by each debugger command.

Chapter 3, “Library Functions”

lists the prototype for each function contained in the libraries provided by the SAS/C Development System. It also gives a brief description of each function.

Conventions

This section covers the typographical and syntax conventions as well as the book codes that reference the books in the SAS/C Development System used in this book.

Typographical Conventions

This book uses special fonts to depict specific types of information. These typographical conventions are as follows:

- `roman` is the basic type style used for most text, table headers, and page references.
- `monospace` is used to show example code. Monospace is used also for items specific to the C language, such as the names of functions, header files, and keywords.
- oblique* is used for arguments or variables whose values are supplied by the user; for example, you should enter an appropriate filename when you see *filename*.
- italic* is used for the descriptions within the tables.

Syntax Conventions

This book uses the following conventions for syntax:

monospace	indicates commands, keywords, and switches that should be spelled exactly as shown. These arguments may or may not be optional, depending on whether they are enclosed in square brackets.
oblique	indicates arguments for which you supply a value.
[] (square brackets)	indicate an optional argument when they surround the argument.
. . . (ellipsis)	indicates that you can repeat the argument or group of arguments preceding the ellipsis any number of times.
(vertical bar)	means to choose one item from a group of items separated by the bars.

The following example illustrates some of these syntax conventions:

env [*function* | *integer*]

env

is a command name, so it appears in monospace type.

function

is a function for which you supply the name, so it appears in oblique type.

[*function* | *integer*]

are both optional, so they are enclosed in square brackets.

function | *integer*

indicates that you can specify only one of the items separated by the vertical bar.

Book Codes

The following book codes are used in this book to reference the library of books in the SAS/C Development System:

- V1 *SAS/C Development System User's Guide, Volume 1: Introduction, Compiler, Editor, Version 6.0*
- V2 *SAS/C Development System User's Guide, Volume 2: Debugger, Utilities, Assembler, Version 6.0*
- LR *SAS/C Development System Library Reference, Version 6.0*

1 Compiler Options

The tables in this chapter list each of the compiler options. For each option, these tables show the following information:

- ☐ the full name.
- ☐ whether the option takes a parameter. An option that takes a parameter is listed with an equals sign (=).
- ☐ the minimum abbreviation for the option.
- ☐ whether the option has a negative form.
- ☐ the default value, if any.
- ☐ a brief description of the option.
- ☐ the name of the scoptions screen on which you can set the option.
- ☐ the page number in *SAS/C Development System User's Guide, Volume 1: Introduction, Editor, Compiler* where you can find a complete description of the option.

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	scoptions Screen	Page Ref.
AbsFuncPointer <i>Generates 32-bit references to functions when loading function pointers</i>	afp	Yes	noafp	Code	V1-85
AddSymbols <i>Tells the linker to add symbol information to the executable module</i>	addsym	Yes	noaddsym	Linker	V1-85
ANSI <i>Enforces the strictest interpretation of the ANSI C standard</i>	ansi	Yes	noansi	Message	V1-85
ArgumentSize= <i>Sets the size of the maximum argument to a preprocessor macro</i>	argsiz	No	512		V1-85
Assembler= <i>Specifies assembly language files that are to be assembled and, if you specify the link option, linked into the program</i>	asm	No			V1-86
AutoRegister <i>Enables automatic register selection by the code generator</i>	autoreg	Yes	autoreg	Code	V1-86

(continued)

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	scoptions Screen	Page Ref.
Batch <i>Tells the linker not to prompt for definitions of undefined symbols</i>	batch	Yes	nobatch	Linker	V1-86
BSSMemory= <i>Specifies the type of memory into which uninitialized external data should be loaded: any, chip, or fast</i>	bssmem	No	any	Code	V1-86
BSSName= <i>Names the uninitialized far data section</i>	bssname	Yes	udata	Code	V1-87
Code= <i>Specifies 16-bit or 32-bit references to functions not declared in the current file: near or far</i>	code	No	near	Code	V1-87
CodeMemory= <i>Specifies the type of memory into which code should be loaded: any, chip, or fast</i>	codemem	No	any	Code	V1-87
CodeName= <i>Names the code section</i>	codename	Yes	text	Code	V1-87
CommentNest <i>Allows nested comments</i>	cnest	Yes	nocnest	Compiler	V1-87
Common <i>Specifies the common model for external data (instead of the strict reference-definition model)</i>	common	Yes	noccommon	Code	V1-88
ConstLibBase <i>Specifies that library base pointers are set once and remain constant throughout your entire program</i>	constlib	Yes	constlib	Code	V1-88
Coverage <i>Generates code that collects coverage analysis information that you can analyze with the cover utility</i>	cover	Yes	nocover	Code	V1-88
CPU= <i>Generates code for the specified processor: any, 68010, 68020, 68030, or 68040</i>	cpu	No	any	Code	V1-89
CSource= <i>Specifies the C source files to be compiled and, if you specify the link option, linked into the program</i>	csrc	No			V1-89

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	scoptions Screen	Page Ref.
Data= <i>Specifies whether you want the compiler to generate 16-bit or 32-bit references to external and static data: near, far, faronly, or auto</i>	data	No	near	Code	V1-89
DataMemory= <i>Specifies the type of memory into which initialized static or external data should be loaded: any, chip, or fast</i>	datamem	No	any	Code	V1-90
DataName= <i>Names the initialized data section</i>	dataname	Yes	data	Code	V1-91
Debug= <i>Sets the debugging level of the compiler: line, symbol, symbolflush, full, or fullflush</i>	dbg	Yes	nodebug	Compiler	V1-91
Define= <i>Defines the specified preprocessor symbol, as if with a #define statement</i>	def	No		Compiler	V1-92
DisAssemble= <i>Disassembles the code as it is generated and sends the disassembly to the file you specify</i>	disasm	Yes	nodisasm	Code	V1-92
Error= <i>Treats the specified message(s) as an error</i>	err	No		Message	V1-92
ErrorConsole <i>Prints diagnostic messages to the console stdout</i>	errcon	Yes	errcon	Message	V1-92
ErrorHighlight <i>Highlights the token that caused the error using ANSI escape sequences in diagnostic output that is sent to the console</i>	errhigh	Yes	errhigh	Message	V1-93
ErrorListing <i>Prints diagnostic messages to the listing file if you also specify the list option</i>	errlist	Yes	errlist	Message	V1-93
ErrorRexx <i>Sends diagnostic message to the scmsg utility</i>	errrexx	Yes	errrexx	Message	V1-93
ErrorSource <i>Prints lines from the C source file with the diagnostic messages that are sent to the console</i>	errsrc	Yes	errsrc	Message	V1-93

(continued)

4 Chapter 1

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	scoptions Screen	Page Ref.
ExternalDefs <i>Treats all external definitions as definitions instead of just declarations</i>	extdef	Yes	extdef		V1-93
FindSymbol= <i>Prints a warning message each time the specified symbol is defined</i>	fsym	No			V1-93
GenProtoDataItems <i>Generates external declarations for variables defined in the source files</i>	gpdata	Yes	gpdata	Prototype	V1-94
GenProtoExterns <i>Generates prototypes for externally known routines if you also specify the genprotos option</i>	gpext	Yes	gpext	Prototype	V1-94
GenProtoFile= <i>Specifies the name of the file into which to place the prototypes generated by the genprotos option</i>	gpfile	Yes	filename_protos.h	Prototype	V1-94
GenProtoParameters <i>Generates prototypes using the __PARMS macro if you also specify the genprotos option</i>	gpparm	Yes	nogpparm	Prototype	V1-94
GenProtos <i>Generates prototypes and data declarations instead of compiling your file</i>	gproto	Yes	nogproto	Prototype	V1-94
GenProtoStatics <i>Generates prototypes for static routines if you also specify the genprotos option</i>	gpstat	Yes	nogpstat	Prototype	V1-95
GenProtoTypeDefs <i>Tells the compiler to use typedefs instead of resolved types when generating prototypes for any functions using typedefs for parameters or return values, if you also specify the genprotos option</i>	gptdef	Yes	gptdef	Prototype	V1-95
GlobalSymbolTable= <i>Tells the compiler to use the specified GST, which must have been created during a previous compilation</i>	gst	Yes	nogst	Compiler	V1-96
GSTImmediate <i>Makes the contents of the GST you specify with the gst option immediately available to the program</i>	gstimm	Yes	nogstimm		V1-96
Icons <i>Tells the compiler to create icons for files that it creates, including listing files, preprocessor output files, prototype files, and GST files</i>	icon	Yes	icons	Compiler	V1-96

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	soptions Screen	Page Ref.
IdentifierLength= <i>Specifies the maximum number of significant characters in an identifier</i>	idlen	No	255		V1-97
Ignore= <i>Tells the compiler to ignore the specified warning message(s)</i>	ign	No		Message	V1-97
IncludeDirectory= <i>Adds the specified directory to the list of directories that you want the compiler to search for #include files</i>	idir	No		Compiler	V1-97
KeepLine <i>Generates #line directives in the preprocessor output file that correspond to the lines in the original source files</i>	kline	Yes	nokline		V1-97
LibCode <i>Tells the compiler that the compiled code will be linked into a shared library</i>	libcode	Yes	nolibcode	Code	V1-98
Library= <i>Specifies the link libraries that are to be passed to the linker</i>	lib	No		Linker	V1-98
Link <i>Tells the compiler to invoke the linker to produce a final executable</i>	link	Yes	nolink	Linker	V1-98
LinkerOptions= <i>Passes the specified parameter(s) to the linker as command line options</i>	linkopt	Yes	nolinkopt	Linker	V1-98
List <i>Tells the compiler or assembler to produce a listing file</i>	list	Yes	nolist	List/Xref	V1-99
ListFile= <i>Names the listing and cross-reference file if you also specify the list option</i>	lfile	Yes	filename.lst	List/Xref	V1-99
ListHeaders <i>Tells the compiler or assembler to include user header files in the listing if you also specify the list option</i>	lhead	Yes	nohead	List/Xref	V1-99
ListIncludes <i>Lists the names of all header files that were included in the program in the listing file if you also specify the list option</i>	linc	Yes	listinc	List/Xref	V1-99

(continued)

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	soptions Screen	Page Ref.
ListMacros <i>Tells the compiler or assembler to expand macros in the listing if you also specify the list option</i>	lmac	Yes	nomac	List/Xref	V1-100
ListNarrow <i>Tells the compiler or assembler to produce a narrow listing (less than 80 columns wide) if you also specify the list option</i>	lnarr	Yes	lnarr	List/Xref	V1-100
ListSystem <i>Tells the compiler to include system header files in the listing if you also specify the list option</i>	lsys	Yes	nolsys	List/Xref	V1-100
MakeGlobalSymbolTable= <i>Creates a Global Symbol Table with the filename that you specify</i>	mgst	Yes		Compiler	V1-100
Map <i>Produces a map of the executable module if you also specify the link option</i>	map	Yes	nomap	Map	V1-100
MapFile= <i>Names the map file</i>	mfile	Yes	executable.map	Map	V1-100
MapHunk <i>Maps all output hunks by size and originating function if you also specify the map option</i>	mhunk	Yes	nomaphunk	Map	V1-101
MapLib <i>Generates a list of hunks by library symbol in the map if you also specify the map option</i>	mlib	Yes	nomlib	Map	V1-101
MapOverlay <i>Includes a list of hunks in each overlay in the map if you also specify the map option</i>	movly	Yes	nomovly	Map	V1-101
MapSymbols <i>Includes a list of defined symbols and the location at which they are defined if you also specify the map option</i>	msym	Yes	nomsym	Map	V1-101
MapXreference <i>Writes a symbol cross-reference to the map file that lists each symbol definition and the places each symbol is used if you also specify the map option</i>	mxref	Yes	nomxref	Map	V1-101
Math= <i>Specifies a format for floating-point math (standard, ffp, 68881, or ieee) and, if you also specify the link option, links with the appropriate math library</i>	math	Yes	nomath	Code	V1-102

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	scoptions Screen	Page Ref.
MaximumErrors= <i>Sets the limit on the number of errors for a single compilation</i>	maxerr	Yes	50	Message	V1-102
MaximumWarnings= <i>Sets the limit on the number of warnings for a single compilation</i>	maxwrn	Yes	nomaxwrn	Message	V1-102
MemorySize= <i>Tells the compiler approximately how much memory you have on your system: tiny, small, medium, large, huge</i>	mensize	No	large	Compiler	V1-103
Modified <i>Tells the sc command to process only files that are out of date with respect to their output files</i>	mod	Yes	nomod	Compiler	V1-104
MultipleCharacterConstants <i>Allows up to four bytes to appear within single quotes as a character constant</i>	mcons	Yes	nomcons		V1-104
MultipleIncludes <i>Tells the compiler to include header files each time they are #included in your program</i>	minc	Yes	minc	Compiler	V1-105
Object= <i>Specifies any object file(s) created during a previous compilation that you want linked into your executable</i>	obj	No		Linker	V1-105
ObjectLibrary= <i>Indicates that any resulting object modules are to be placed into the specified link library</i>	objlib	Yes		Code	V1-106
ObjectName= <i>Specifies the name of the file (if processing one source file) or directory (if processing more than one file) to hold compiler and assembler output</i>	objname	Yes		Code	V1-106
OldPreProcessor <i>Allows pre-ANSI style token passing and substitutes values for parameters specified as quoted strings in macro definitions</i>	oldpp	Yes	nooldpp		V1-106
OnError= <i>Tells sc what action to take if a C language or assembly language source file generates an error: stop or continue</i>	onerr	No	stop	Message	V1-107

(continued)

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	soptions Screen	Page Ref.
Optimize <i>Enables the global optimizer (unless you specify nooptglobal) and peephole optimizer (unless you specify nooptpeep)</i>	opt	Yes	noopt	Optimizer	V1-108
OptimizerAlias <i>Disables type-based aliasing assumptions in the global optimizer (uses worst-case aliasing)</i>	optalias	Yes	optalias	Optimizer	V1-108
OptimizerComplexity= <i>Defines the maximum complexity level of functions to be automatically inlined by the global optimizer (the number of discrete operations in the functions)</i>	optcomp	Yes	3	Optimizer	V1-108
OptimizerDepth= <i>Defines the maximum nesting depth of automatically inlined functions in the global optimizer</i>	optdep	Yes	3	Optimizer	V1-109
OptimizerGlobal <i>Enables the global optimizer</i>	optgo	Yes	optgo	Optimizer	V1-109
OptimizerInline <i>Allows inlining of functions by the global optimizer, including functions defined with the <code>__inline</code> keyword</i>	optinl	Yes	optinl	Optimizer	V1-109
OptimizerInlocal <i>Inlines single-use static functions in the global optimizer</i>	optinlocal	Yes	nooptinlocal	Optimizer	V1-109
OptimizerLoop <i>Enables loop optimizations by the global optimizer</i>	optloop	Yes	optloop	Optimizer	V1-109
OptimizerPeephole <i>Enables the peephole optimizer</i>	optpeep	Yes	optpeep	Optimizer	V1-109
OptimizerRecurDepth= <i>Defines the maximum depth of recursion of automatically inlined functions by the global optimizer</i>	optrdep	Yes	3	Optimizer	V1-110
OptimizerSize <i>Disables optimizations that may sacrifice code size to save time</i>	optsize	Yes	nooptsize	Optimizer	V1-110
OptimizerTime <i>Disables optimizations that may sacrifice time to save space</i>	opttime	Yes	noopttime	Optimizer	V1-110

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	scoptions Screen	Page Ref.
Parameters= <i>Indicates how parameters should be passed: stack, register, or both</i>	parm	No	stack	Code	V1-110
Precision= <i>Specifies the size of floating-point variables</i>	prec	No	mixed	Code	V1-111
PreProcessorBuffer= <i>Sets the maximum number of bytes to which a macro can be expanded by the preprocessor</i>	ppbuf	No	8192		V1-111
PreProcessorOnly <i>Tells the compiler to run only the preprocessor on the C source files</i>	pponly	Yes	nopponly	Compiler	V1-111
ProgramName= <i>Specifies the name of the executable module</i>	pname	No		Index	V1-112
ResetOptions <i>Resets all options preceding it on the command line (including any options read from an scoptions file) to their default values</i>	resopt	No			V1-112
Saveds <i>Generates code as if you had defined all functions in the source files with the __saveds keyword</i>	saveds	Yes	nosaveds	Code	V1-112
ShortIntegers <i>Enables 16-bit integers</i>	sint	Yes	nosint	Compiler	V1-112
SmallCode <i>Tells the linker to merge all code hunks into a single hunk</i>	scode	Yes	noscode	Linker	V1-113
SmallData <i>Tells the linker to merge all data and BSS sections into a single hunk</i>	sdata	Yes	nosdata	Linker	V1-113
SourceIs= <i>Sets the name of the C source file in the object file and in debugging information to the specified value instead of the actual name of the C source file</i>	srcis	Yes		Code	V1-113
StackCheck <i>Generates stack overflow checking code at each function entry</i>	stkchk	Yes	stkchk	Code	V1-113

(continued)

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	scoptions Screen	Page Ref.
StackExtend <i>Generates stack extension code at each function entry</i>	stkext	Yes	nostkext	Code	V1-113
StandardIO <i>Specifies the use of __main instead of __tinymain</i>	stdio	Yes	stdio	Linker	V1-114
Startup= <i>Specifies which startup module to use</i>	strt	Yes	c (for c.o)	Linker	V1-114
Strict <i>Enables large number of diagnostics dealing with portability and questionable situations in your code</i>	strict	Yes	nostrict	Message	V1-114
StringsConst <i>Tells the compiler to consider all string constants to be of type const char *</i>	strcon	Yes	nostrcons	Compiler	V1-114
StringMerge <i>Merges all identical string constants in the C source file and places all strings, all data declared static const, and all auto initializers in the code section instead of the data section</i>	strmer	Yes	nostrmer	Compiler	V1-114
StripDebug <i>Strips all debugging information from the final executable</i>	stripdbg	Yes	nostripdbg	Linker	V1-116
StructureEquivalence <i>Tells the compiler not to issue messages if a pointer to one structure type is passed to a function when the function expects a pointer to a different type, if the type passed is equivalent to the type expected</i>	streq	Yes	nostrreq	Message	V1-116
Trigraph <i>Specifies whether to use ANSI trigraphs</i>	trig	Yes	notrig		V1-116
Underscore <i>Adds an underscore to the beginning of all external names in any assembler source files assembled</i>	uscore	Yes	nouscore		V1-116
UnsignedChar <i>Makes the default type of char variables unsigned instead of signed</i>	uchar	Yes	nouchar	Compiler	V1-116

Option Name and Description	Minimum Abbreviation	Negative Form?	Default Value	scoptions Screen	Page Ref.
UtilityLibrary <i>Generates inline calls to the AmigaDOS 2.0 ROM-resident library utility.library to do integer multiplication and division instead of calling SAS/C library functions to do these operations</i>	utllib	Yes	noutllib	Code	V1-117
Verbose <i>Displays messages about each stage of compiling and linking</i>	verbose	Yes	noverbose	Index	V1-117
Version <i>Prints a banner containing the compiler version number and a copyright message</i>	ver	Yes	version	Index	V1-117
Warn= <i>Enables the specified compiler warning message(s)</i>	wrn	No		Message	V1-117
WarnVoidReturn <i>Issues a warning message if a function declared as returning an integer actually returns no value</i>	wvret	Yes	wvret	Compiler	V1-118
With= <i>Specifies the name of a file containing additional options</i>	with	No			V1-118
Xreference <i>Produces a cross-reference</i>	xref	Yes	noxref	List/Xref	V1-118
XreferenceHeaders <i>Generates a cross-reference of user header files if you also specify the xreference option</i>	xhead	Yes	noxhead	List/Xref	V1-118
XreferenceSystem <i>Includes symbols defined in system header files in the cross-reference if you also specify the xreference option</i>	xsys	Yes	noxsys	List/Xref	V1-119

2 Debugger Commands

The tables in this chapter list each of the commands available in the SAS/C Debugger. For each command, these tables show the following information:

- ☐ the command name
- ☐ a brief description of the command
- ☐ any options available with the command
- ☐ any parameters required by the command or its options
- ☐ the page number in *SAS/C Development System User's Guide, Volume 2: Debugger, Utilities, Assembler* where you can find a complete description of the command.

Command Name	Description and Syntax	Page Ref.
activate	<i>Activates a task under debugger control</i> a[ctivate] [task-name task-address exe-name]	V2-127
alias	<i>Defines an alias for a debugger command</i> al[ias] [name[definition]]	V2-128
args	<i>Displays the arguments to a function</i> ar[gs] [function-name]	V2-130
bclear	<i>Clears one or more breakpoints</i> bc[lear] integer [integer ...] bc[lear] * l[ast] bc[lear] integer .. integer	V2-131
bdisable	<i>Disables one or more breakpoints</i> bd[isable] integer[[integer ...] bd[isable] * l[ast] bd[isable] integer w .. integer	V2-132
benable	<i>Enables one or more breakpoints</i> be[enable] integer[integer ...] be[enable] * l[ast] be[enable] integer .. integer	V2-133

(continued)

Command Name	Description and Syntax	Page Ref.
blist	<i>Lists all breakpoints</i> bl[ist]	V2-134
break	<i>Sets a breakpoint</i> b[reak] location[after(integer)][when(expression)] [tr[ace]] [q[uiet]] [te[mp]] [{cmd-list}]	V2-135
call	<i>Evaluates an expression and discards the result</i> c[all] expression	V2-137
catch	<i>Catches a task not currently under debugger control</i> ca[tch] [task-name address exe-name]	V2-139
deactivate	<i>Deactivates a task under debugger control</i> dea[ctivate] [task-name address exe-name]	V2-140
define	<i>Defines a macro</i> {#} def[ine] name [definition] {#} def[ine] name(parm [,parm ...]) [definition]	V2-141
detach	<i>Detaches a task from debugger control</i> det[ach][task-name address exe-name]	V2-142
display	<i>Displays the value of an expression</i> d[isplay] (expression["format"] string) [, ...]	V2-143
dump	<i>Dumps memory contents</i> du[mp] [variable address range] [\$] [format [itemsizes] text]	V2-145
dzero	<i>Displays memory as a null-terminated ASCII string</i> dz[ero] variable address array-slice	V2-148
echo	<i>Displays a string</i> ec[ho] [text]	V2-149
env	<i>Sets the environment</i> env [function integer] env [-c[aller]] -s[ubroutine] -u[p] -d[own] -integer +integer]	V2-150

Command Name	Description and Syntax	Page Ref.
execute	<i>Executes a debugger command file</i> ex[ecute] filename	V2-152
expand	<i>Expands and displays a command line</i> expand [alias] command-line	V2-153
finish	<i>Terminates CPRK and CPRX</i> fin[ish]	V2-154
fregister	<i>Displays or modifies floating-point registers</i> fr[egister] [register[[=] expression]]	V2-155
go	<i>Continues execution until a breakpoint is encountered or the program exits</i> g[o] [location[after(integer)] [when(expression)]]	V2-156
help	<i>Displays help information</i> h[elp] [command] ? [command]	V2-158
hunks	<i>Lists the addresses and sizes of all hunks</i> hu[nks]	V2-159
jump	<i>Changes the current execution point</i> j[ump] location	V2-160
list	<i>Lists the lines in a source file</i> l[list] l[list] \$ l[list] + -integer [L length] l[list] line-range l[list] [image:] \module [line-range] l[ist] [[image:] \module \]function [line-range]	V2-161
log	<i>Logs debugger commands to a file</i> lo[g] [log-command]	V2-163

(continued)

Command Name	Description and Syntax	Page Ref.
opt	<i>Shows option values or changes the value of a debugger option</i> op[t] op[t] au[toswap] [on off] op[t] ar[rdim] integer op[t] b[adchar] integer op[t] ca[se] [on off] op[t] cat[ch] [on off] op[t] co[ntext] integer op[t] dev[ices] [on off] op[t] e[cho] [on off] op[t] ib[ytes] [on off] op[t] ig[norepath] [on off] op[t] l[ist] integer op[t] rad[ix] {d[ecimal] h[ex]} op[t] ran[gelen] integer op[t] re[slib] [on off] op[t] se[arch] [+ -] directory [, directory [...]] op[t] so[urce] c a[as] m[ixed] n[ext] op[t] st[rlen] integer op[t] t[ask] [task-ID] op[t] u[nassemble] integer	V2-165
proceed	<i>Single-steps over function calls</i> p[roceed] [integer]	V2-168
ps	<i>Single-steps over function calls by source line</i> ps [integer]	V2-169
quit	<i>Terminates the debugger</i> q[uit]	V2-170
register	<i>Displays or modifies register</i> r[egister] [register [=] expression]	V2-171
restart	<i>Restarts the program being debugged</i> res[tart] [argument-list]	V2-172
return	<i>Returns immediately from the current function</i> ret[urn] [expression]	V2-174
rflag	<i>Displays or modifies flags</i> rf[lag] [flag ...]	V2-175

Command Name	Description and Syntax	Page Ref.
search	<i>Searches for a string in the current source file</i> sea[rch] [string]	V2-176
set	<i>Modifies the values of variables or memory locations</i> se[t] (variable register) = expression se[t] address[=] string	V2-177
sleep	<i>Pauses for the time specified</i> sle[ep] number	V2-179
source	<i>Displays source file</i> so[urce] [filename]	V2-180
start	<i>Restarts the program being debugged</i> sta[rt] [argument-list]	V2-181
symbol	<i>Finds the symbol nearest to the specified address</i> symb[ol] address	V2-183
symload	<i>Reads symbol information from an executable file</i> sym[load] executable-file sym[load] proc process-id sym[load] seg address executable-file	V2-184
tasks	<i>Displays system tasks</i> ta[sks] [a[ll]]	V2-185
trace	<i>Single-steps into function calls</i> t[race] [integer]	V2-186
ts	<i>Single-steps by source line into function calls</i> ts [integer]	V2-187
unalias	<i>Deletes an alias</i> una[lias] name una[lias] *	V2-188
unassemble	<i>Displays memory as assembler instructions</i> u[nassemble] [location-1[location-2]]	V2-189

(continued)

Command Name	Description and Syntax	Page Ref.
undefine	<i>Deletes a macro definition</i> [#]und[efine] name [#]und[efine] *	V2-191
watch	<i>Sets a watch on a variable or memory</i> w[atch] expression range[s[tatic] d[ynamic]]	V2-192
wbreak	<i>Sets a watch break</i> wb[reak] expression range [s[tatic] d[ynamic]]	V2-193
wclear	<i>Clears one or more watches</i> wc[lear] integer[integer ...] wc[lear] * l[ast] wc[lear] integer .. integer	V2-194
wdisable	<i>Disables one or more watches</i> wd[isable] integer[integer ...] wd[isable] * l[ast] wd[isable] integer .. integer	V2-195
wenable	<i>Enables one or more watches</i> we[nable] integer ... we[nable] [*] l[ast] we[nable] integer .. integer	V2-196
whatis	<i>Determines the type of an object</i> wha[tis] expression wha[tis] {type}	V2-197
where	<i>Shows the calling sequence</i> whe[re] [a[rgs]] [integer]	V2-198
window	<i>Opens or closes a window</i> wi[ndow] window-name {on off}	V2-199
wlist	<i>Lists all watches</i> wl[ist]	V2-200
wmsg	<i>Writes a message to the Message window</i> wmsg integer message-text	V2-201

Code Probe Built-in Functions

The following list provides syntax information for the built-in functions that are included with the CodeProbe debugger:

Command Name	Description and Syntax	Page Ref.
memcmp	<i>Compares two memory blocks</i> <pre>i = memcmp(a, b, n); int i; /* comparison results */ void *a, *b; /* blocks being compared */ int n; /* block size in bytes */</pre>	V2-204
memcpy	<i>Copies a memory block (non-overlapping)</i> <pre>to = memcpy(to, from, n); void *to; /* destination pointer */ void *from; /* source pointer */ int n; /* block size in bytes */</pre>	V2-205
memmove	<i>Copies a memory block (possibly overlapping)</i> <pre>to = memmove(to, from, n); void *to; /* destination pointer */ void *from; /* source pointer */ int n; /* block size in bytes */</pre>	V2-206
memset	<i>Sets memory to specified value</i> <pre>to = memset(to, c, n); void *to /* base of memory to be initialized */ int c; /* initialization value */ int n; /* number of bytes to be initialized */</pre>	V2-207
strcat	<i>Concatenates two strings</i> <pre>to = strcat(to, from); char *to; /* destination pointer */ char *from; /* source pointer */</pre>	V2-208
strcmp	<i>Compares two strings</i> <pre>i = strcmp(a, b); int i; /* comparison result */ char *a, *b; /* strings being compared */</pre>	V2-209

(continued)

Command Name	Description and Syntax	Page Ref.
strcpy	<i>Copies a string</i> to = strcpy(to, from); char *to; /* destination pointer */ char *from; /* source pointer */	V2-210
strlen	<i>Returns the length of a string</i> len = strlen(s); int len; /* length of string s */ char *s; /* string to scan for length */	V2-211

3 Library Functions

The tables in this chapter list each of the library functions available in the SAS/C Libraries. For each function, these tables show the following information:

- ☐ the function name
- ☐ the header file containing the prototype of the function
- ☐ a brief description of the function
- ☐ the function's prototype
- ☐ the page number in *SAS/C Development System Library Reference* where you can find a complete description of the function.

Function Name	Header File	Description and Prototype	Page Ref.
abort	stdlib.h	<i>Abort the current process</i> void abort(void);	LR-103
abs	stdlib.h	<i>Absolute value</i> type abs(type x);	LR-104
access	stdio.h	<i>Check file accessibility</i> int access(const char *name, int mode);	LR-105
acos	math.h	<i>Arccosine function</i> double acos(double x);	LR-107
argopt	stdlib.h	<i>Get options from an argument list</i> char *argopt(int argc, const char **argv, const char *opts, int *argn, char *optc);	LR-108
asctime	time.h	<i>Generate an ASCII time string</i> char *asctime(const struct tm *t);	LR-111
asin	math.h	<i>Arcsine function</i> double asin(double x);	LR-113
assert, __assert	assert.h	<i>Assert program validity</i> void assert(int x); void __assert(int exp, const char *file, int *line);	LR-114

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
astcsma	string.h	<i>AmigaDOS string pattern match (anchored)</i> int astcsma(const char *s, const char *p);	LR-116
atan	math.h	<i>Arctangent function</i> double atan(double x);	LR-118
atan2	math.h	<i>Arctangent of x/y</i> double atan2(double x, double y);	LR-119
atexit	stdlib.h	<i>Set an exit trap</i> int atexit(void (*func)(void));	LR-120
atof	stdlib.h	<i>Convert an ASCII string to a float-point number</i> double atof(const char *p);	LR-121
atoi	stdlib.h	<i>Convert an ASCII string to an integer</i> int atoi(const char *s);	LR-123
atol	stdlib.h	<i>Convert an ASCII string to a long integer</i> long int atol(const char *s);	LR-124
bldmem	stdlib.h	<i>Build a memory pool of specified size</i> int bldmem(int n);	LR-125
bsearch	stdlib.h	<i>Perform a binary search</i> void *bsearch(const void *srch, const void *array, size_t n, size_t size, int (*cmp)(const void *, const void *));	LR-126
calloc	stdlib.h	<i>Allocate and clear memory</i> void *calloc(size_t nelt, size_t esize);	LR-127
ceil	math.h	<i>Get floating-point limits</i> double ceil(double y);	LR-129
chdir	stdio.h	<i>Change the current directory</i> int chdir(const char *path);	LR-130
chgclk	dos.h	<i>Change the system clock</i> int chgclk(const unsigned char *clock);	LR-131
Chk Abort, chkabort	dos.h	<i>Check for a break character</i> void Chk_Abort(void); void chkabort(void);	LR-132

Function Name	Header File	Description and Prototype	Page Ref.
chkml	stdlib.h	<i>Check for the largest memory block</i> long chkml(void);	LR-133
chkufb	ios1.h	<i>Check a level 1 file handle</i> struct UFB *chkufb(int fh);	LR-134
chmod	stdio.h fcntl.h	<i>Change a file's protection mode</i> int chmod(const char *name, int mode);	LR-135
clearerr	stdio.h	<i>Clear a level 2 I/O error flag</i> void clearerr(FILE *fp);	LR-137
clock	time.h	<i>Measure program processor time</i> clock_t clock(void);	LR-138
close	fcntl.h	<i>Close a level 1 file</i> int close (int fh);	LR-140
closedir	dir.h	<i>Terminate the directory operation</i> void closedir(DIR *dfd);	LR-141
clrerr	stdio.h	<i>Clear a level 2 I/O error flag</i> void clrerr(FILE *fp);	LR-142
cos	math.h	<i>Cosine function</i> double cos(double x);	LR-143
cosh	math.h	<i>Hyperbolic cosine function</i> double cosh(double x);	LR-144
cot	math.h	<i>Cotangent function</i> double cot(double x);	LR-145
creat	fcntl.h	<i>Create a level 1 file</i> int creat(const char *name, int prot);	LR-146
ctime	time.h	<i>Convert a time_t value to an ASCII string</i> char *ctime(const time_t *t);	LR-147
__ctype	ctype.h	<i>Character class table</i> char __ctype[];	LR-45

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
<code>_CXFERR</code>	<code>math.h</code>	<i>Low-level float error</i> <code>void _CXFERR(int code);</code>	LR-149
<code>_CXOVF</code>		<i>Default stack-overflow handler</i> <code>void _CXOVF(void);</code>	LR-150
<code>datecmp</code>	<code>dos.h</code>	<i>Compare two AmigaDOS dates</i> <code>int datecmp(const struct DateStamp *d1, const struct DateStamp *d2);</code>	LR-151
<code>_dclose</code>	<code>dos.h</code>	<i>Close an AmigaDOS file</i> <code>int _dclose(long fh);</code>	LR-152
<code>_dcreat</code>	<code>dos.h</code>	<i>Create an AmigaDOS file</i> <code>long _dcreat(const char *name, int fatt);</code>	LR-153
<code>_dcreatx</code>	<code>dos.h</code>	<i>Create a new AmigaDOS file</i> <code>long _dcreatx(const char *name, int fatt);</code>	LR-154
<code>dfind</code>	<code>dos.h</code>	<i>Find a directory entry</i> <code>int dfind(struct FileInfoBlock *info, const char *name, int attr);</code>	LR-155
<code>difftime</code>	<code>time.h</code>	<i>Compute the difference between two time_t values</i> <code>double difftime(time_t time2, time_t time1);</code>	LR-157
<code>div</code>	<code>stdlib.h</code>	<i>Compute the quotient and the remainder</i> <code>div_t div(int y, int x);</code>	LR-159
<code>dnext</code>	<code>dos.h</code>	<i>Find the next directory entry</i> <code>int dnext(struct FileInfoBlock *info);</code>	LR-161
<code>_dopen</code>	<code>dos.h</code>	<i>Open an AmigaDOS file</i> <code>long _dopen(const char *name, int mode);</code>	LR-163
<code>dqsort</code>	<code>stdlib.h</code>	<i>Sort an array of doubles</i> <code>void dqsort(double *da, size_t n);</code>	LR-164
<code>drand48</code>	<code>math.h</code>	<i>Generate a random double (internal seed)</i> <code>double drand48(void);</code>	LR-165
<code>dread</code>	<code>dos.h</code>	<i>Read an AmigaDOS file</i> <code>unsigned int _dread(long fh, char *buffer, unsigned int length);</code>	LR-166

Function Name	Header File	Description and Prototype	Page Ref.
_dseek	dos.h	<i>Reposition an AmigaDOS file</i> long _dseek(long fh, long rpos, int mode);	LR-167
_dwrite	dos.h	<i>Write to an AmigaDOS file</i> unsigned int _dwrite(long fh, char *buffer, unsigned int length);	LR-168
ecvt	math.h	<i>Convert a double to a string</i> char *ecvt(double v, int dig, int *decx, int *sign);	LR-169
__emit	dos.h	<i>Emit 68000 instruction word</i> void __emit(int inst);	LR-171
erand48	math.h	<i>Generate a random double (external seed)</i> double erand48(unsigned short *seed);	LR-172
except	math.h	<i>Call the math error handler function</i> double except(int type, const char *name, double arg1, double arg2, double retval);	LR-173
exit	stdlib.h	<i>Terminate program execution</i> void exit(int code);	LR-175
__exit	stdlib.h	<i>Terminate program execution with no clean-up</i> void __exit(int code);	LR-177
exp	math.h	<i>Exponential function</i> double exp(double x);	LR-178
fabs	math.h	<i>Float or double absolute value</i> double fabs(double d);	LR-179
fclose	stdio.h	<i>Close a level 2 file</i> int fclose(FILE *fp);	LR-180
fcloseall	stdio.h	<i>Close all level 2 files</i> int fcloseall(void);	LR-181
fcvt	math.h	<i>Convert a float to a string</i> char *fcvt(double v, int dig, int *decx, int *sign);	LR-182

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
fopen	stdio.h	<i>Attach a level 1 file to level 2</i> <code>FILE *fopen(int fh, const char *mode);</code>	LR-184
feof	stdio.h	<i>Check for a level 2 end-of-file</i> <code>int feof(FILE *fp);</code>	LR-185
ferror	stdio.h	<i>Check for a level 2 error</i> <code>int ferror(FILE *fp);</code>	LR-186
fflush	stdio.h	<i>Flush a level 2 output buffer</i> <code>int fflush(FILE *fp);</code>	LR-187
fgetc	stdio.h	<i>Get a character from a file</i> <code>int fgetc(FILE *fp);</code>	LR-188
fgetchar	stdio.h	<i>Get a character from stdin</i> <code>int fgetchar(void);</code>	LR-189
fgetpos	stdio.h	<i>Get the current file position</i> <code>int fgetpos(FILE *f, fpos_t *pos);</code>	LR-190
fgets	stdio.h	<i>Get a string from a level 2 file</i> <code>char *fgets(char *buffer, int length, FILE *fp);</code>	LR-193
fileno	stdio.h	<i>Get the file number for a level 2 file</i> <code>int fileno(FILE *fp);</code>	LR-195
findpath	dos.h	<i>Locate a file in the current path</i> <code>BPTR findpath(const char *filename);</code>	LR-196
floor	math.h	<i>Get the floor of a real number</i> <code>double floor(double y);</code>	LR-198
flushall	stdio.h	<i>Flush all level 2 output buffers</i> <code>int flushall(void);</code>	LR-199
fmod	math.h	<i>Float modulus operations</i> <code>double fmod(double y, double z);</code>	LR-200
fmode	stdio.h	<i>Change the mode of level 2 file</i> <code>void fmode(FILE *fp, int mode);</code>	LR-201
fopen	stdio.h	<i>Open a level 2 file</i> <code>FILE *fopen(const char *name, const char *mode);</code>	LR-202

Function Name	Header File	Description and Prototype	Page Ref.
forkl	dos.h	<i>Create a child process with an argument list</i> int forkl(char *prog, char *arg0, ..., NULL, struct FORKENV *env, struct ProcID *procid);	LR-207
forkv	dos.h	<i>Create a child process with an argument vector</i> int forkv(char *prog, char **argv, struct FORKENV *env, struct ProcID *procid);	LR-207
fprintf	stdio.h	<i>Formatted print</i> int fprintf(FILE *fp, const char *fmt, ...);	LR-214
fputc	stdio.h	<i>Put a character to a level 2 file</i> int fputc(int c, FILE *fp);	LR-221
fputchar	stdio.h	<i>Put a character to stdout</i> int fputchar(int c);	LR-222
fputs	stdio.h	<i>Put a string to a level 2 file</i> int fputs(const char *s, FILE *fp);	LR-223
fqsort	stdlib.h	<i>Sort an array of floats</i> void fqsort(float *fa, size_t n);	LR-224
fread	stdio.h	<i>Read and write blocks</i> size_t fread(void *b, size_t bsize, size_t n, FILE *fp);	LR-225
free	stdlib.h	<i>Free memory</i> void free(void *b);	LR-226
freopen	stdio.h	<i>Reopen a level 2 file</i> FILE *freopen(const char *name, const char *mode, FILE *fp);	LR-229
frexp	math.h	<i>Split a float value</i> double frexp(double v, int *xp);	LR-230
fscanf	stdio.h	<i>Formatted input conversions</i> int fscanf(FILE *fp, const char *fmt, ...);	LR-231
fseek	stdio.h	<i>Set a level 2 file position</i> int fseek(FILE *fp, long int rpos, int mode);	LR-234

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
fsetpos	stdio.h	<i>Reposition a file</i> int fsetpos(FILE *fp, const fpos_t *pos);	LR-235
fstat	stat.h	<i>Get file status</i> int fstat(int file, struct stat *st);	LR-236
ftell	stdio.h	<i>Get a level 2 file position</i> long int ftell(FILE *fp);	LR-238
fwrite	stdio.h	<i>Write blocks to a level 2 file</i> size_t fwrite(const void *b, size_t bsize, size_t n, FILE *fp);	LR-239
gcvt	math.h	<i>Convert a float to a string</i> char *gcvt(double v, int dig, char *buffer);	LR-240
geta4	dos.h	<i>Establish addressability to the global data area</i> void geta4(void);	LR-241
getasn	dos.h	<i>Get assigned device</i> struct DeviceList *getasn(const char *name);	LR-242
getc	stdio.h	<i>Get a character from a file</i> int getc(FILE *fp);	LR-244
getcd	dos.h	<i>Get the current directory</i> int getcd(int drive, char *path);	LR-245
getch	stdio.h	<i>Get a character from stdin immediately</i> int getch(void);	LR-246
getchar	stdio.h	<i>Get a character from stdin</i> int getchar(void);	LR-247
getclk	dos.h	<i>Get or change the system clock</i> void getclk(unsigned char *clock);	LR-248
getcwd	dos.h	<i>Get the current working directory</i> char *getcwd(char *b, int size);	LR-249
getdfs	dos.h	<i>Get disk free space</i> int getdfs(const char *drive, struct InfoData *info);	LR-250

Function Name	Header File	Description and Prototype	Page Ref.
getenv	stdlib.h	<i>Get an environment variable</i> char *getenv(const char *name);	LR-252
getfa	dos.h	<i>Get the file attribute</i> int getfa(const char *name);	LR-254
getfnl	dos.h	<i>Get a filename list</i> int getfnl(const char *fnp, char *fna, size_t fnasize, int attr);	LR-255
getft	dos.h	<i>Get the file time</i> long getft(const char *name);	LR-258
getmem	stdlib.h	<i>Get a memory block (short)</i> void *getmem(unsigned int sbytes);	LR-259
getml	stdlib.h	<i>Get a memory block (long)</i> void *getml(long lbytes);	LR-260
getpath	dos.h	<i>Get the path for a specific directory or file</i> int getpath(BPTR lock, char *path);	LR-261
getreg	dos.h	<i>Get 68000-specific registers</i> long getreg(int reg);	LR-262
gets	stdio.h	<i>Get a string from stdin</i> char *gets(char *buffer);	LR-264
gmtime	time.h	<i>Unpack Greenwich Mean Time (GMT)</i> struct tm *gmtime(const time_t *t);	LR-266
halloc	stdlib.h	<i>Allocate a huge memory block</i> void *halloc(unsigned long n);	LR-268
iabs	stdlib.h	<i>Integer absolute value</i> int iabs(int i);	LR-269
iomode	fcntl.h	<i>Change the mode of a level 1 file</i> int iomode (int fh, int mode);	LR-270
isalnum	ctype.h	<i>Test if a character is alphanumeric</i> int isalnum(int c);	LR-271

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
isalpha	ctype.h	<i>Test if a character is alphabetic</i> <code>int isalpha(int c);</code>	LR-271
isascii	ctype.h	<i>Test if a character is an ASCII character</i> <code>int isascii(int c);</code>	LR-271
isctrl	ctype.h	<i>Test if a character is a control character</i> <code>int isctrl(int c);</code>	LR-271
iscsym	ctype.h	<i>Test if a character is a C symbol character</i> <code>int iscsym(int c);</code>	LR-271
iscsymf	ctype.h	<i>Test if a character is a C symbol lead character</i> <code>int iscsymf(int c);</code>	LR-271
isdigit	ctype.h	<i>Test if a character is a decimal digit (0 to 9)</i> <code>int isdigit(int c);</code>	LR-271
isgraph	ctype.h	<i>Test if a character is a graphic character</i> <code>int isgraph(int c);</code>	LR-271
islower	ctype.h	<i>Test if a character is lowercase</i> <code>int islower(int c);</code>	LR-271
isprint	ctype.h	<i>Test if a character is printable</i> <code>int isprint(int c);</code>	LR-271
ispunct	ctype.h	<i>Test if a character is a punctuation character</i> <code>int ispunct(int c);</code>	LR-271
isspace	ctype.h	<i>Test if a character is a space character</i> <code>int isspace(int c);</code>	LR-271
isupper	ctype.h	<i>Test if a character is uppercase</i> <code>int isupper(int c);</code>	LR-271
isxdigit	ctype.h	<i>Test if a character is a hex character (0-9, A-F, a-f)</i> <code>int isxdigit(int c);</code>	LR-271
isatty	fcntl.h	<i>Test a file descriptor for a terminal device</i> <code>int isatty(int fd);</code>	LR-273
jrand48	math.h	<i>Generate a random long integer (external seed)</i> <code>long jrand48(unsigned short *seed);</code>	LR-274

Function Name	Header File	Description and Prototype	Page Ref.
labs	stdlib.h	<i>Long integer absolute value</i> long int labs(long int l);	LR-275
lcong48	math.h	<i>Set linear congruence parameters</i> void lcong48(unsigned hort *parm);	LR-276
ldexp	math.h	<i>Combine floating-point value</i> double ldexp(double d, int x);	LR-277
ldiv	stdlib.h	<i>Return the long integer quotient and the remainder</i> ldiv_t ldiv(long int numer, long int denom);	LR-278
localeconv	locale.h	<i>Return information on locale formatting conventions</i> struct lconv *localeconv(void);	LR-280
localtime	time.h	<i>Unpack local time</i> struct tm *localtime(const time_t *t);	LR-282
log	math.h	<i>Natural logarithm function</i> double log(double x);	LR-283
log10	math.h	<i>Base 10 logarithm function</i> double log10(double x);	LR-284
longjmp	setjmp.h	<i>Perform a long jump</i> void longjmp(jmp_buf save, int value);	LR-285
lqsort	stdlib.h	<i>Sort an array of long integers</i> void lqsort(long *la, size_t n);	LR-286
lrand48	math.h	<i>Generate a random positive long integer (internal seed)</i> long lrand48(void);	LR-287
lsbrk	stdlib.h	<i>Allocate memory</i> void *lsbrk(long lbytes);	LR-288
lseek	fcntl.h	<i>Set a level 1 file position</i> long lseek(int fh, long rpos, int mode);	LR-289
lstat	stat.h	<i>Get file status</i> int lstat(const char *file, struct stat *st);	LR-291

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
<code>__main</code>	<code>stdlib.h</code>	<i>Standard preprocessing for the main module</i> <code>void __stdargs __main(char *line);</code>	LR-293
<code>main</code>		<i>Your main or principal function</i> <code>int main(int argc, char **argv);</code>	LR-294
<code>malloc</code>	<code>stdlib.h</code>	<i>Allocate memory</i> <code>void *malloc(size_t n);</code>	LR-298
<code>MathBase</code>		<i>FFP Library Vector</i> <code>struct Library *MathBase;</code>	LR-59
<code>__matherr</code>	<code>math.h</code>	<i>Math error handler</i> <code>int __matherr(struct __exception *x);</code>	LR-300
<code>max</code>	<code>math.h</code>	<i>Compute the maximum of two values</i> <code>type max(type a, type b);</code>	LR-302
<code>mblen</code>	<code>stdlib.h</code>	<i>Determine the length of a multibyte character</i> <code>int mblen(const char *s, size_t n);</code>	LR-303
<code>mbstowcs</code>	<code>stdlib.h</code>	<i>Convert a multibyte string to a wide character string</i> <code>size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);</code>	LR-304
<code>mbtowc</code>	<code>stdlib.h</code>	<i>Map a multibyte character to a wide character</i> <code>int mbtowc(wchar_t *pwc, const char *s, size_t n);</code>	LR-305
<code>memccpy</code>	<code>string.h</code>	<i>Copy a memory block</i> <code>void *memccpy(void *to, const void *from, int c, unsigned n);</code>	LR-306
<code>memchr</code>	<code>string.h</code>	<i>Find a character in a memory block</i> <code>void *memchr(const void *a, int c, size_t n);</code>	LR-307
<code>_MemCleanup</code>	<code>stdlib.h</code>	<i>Deallocate all allocated memory</i> <code>void __stdargs _MemCleanup(void);</code>	LR-308
<code>memcmp</code>	<code>string.h</code>	<i>Compare two memory blocks</i> <code>int memcmp(const void *a, const void *b, size_t n);</code>	LR-309
<code>memcpy</code>	<code>string.h</code>	<i>Copy a memory block</i> <code>void *memcpy(void *to, const void *from, size_t n);</code>	LR-310

Function Name	Header File	Description and Prototype	Page Ref.
memmove	string.h	<i>Copy bytes in memory</i> void *memmove(void *dest, const void *from, size_t nbytes);	LR-311
memset	string.h	<i>Set a memory block to a value</i> void *memset(void *to, int c, size_t n);	LR-313
min	math.h	<i>Compute the minimum of two values</i> type min(type a, type b);	LR-314
mkdir	dos.h	<i>Make a new directory</i> int mkdir(const char *path);	LR-315
mktime	time.h	<i>Convert the broken down time to a time_t value</i> time_t mktime(struct tm *ts);	LR-316
modf	math.h	<i>Split a floating-point value</i> double modf(double y, double *p);	LR-318
movmem	string.h	<i>Move a memory block</i> void movmem(const void *from, void *to, unsigned n);	LR-320
rand48	math.h	<i>Generate a random long integer (internal seed)</i> long rand48(void);	LR-321
rand48	math.h	<i>Generate a random positive long integer (external seed)</i> long rand48(unsigned short *seed);	LR-322
offsetof	stddef.h	<i>Get the byte offset of a structure member</i> size_t offsetof(structure-type, member);	LR-323
onbreak	dos.h	<i>Plant a break trap</i> int onbreak(int (*func)(void));	LR-326
onexit	stdlib.h	<i>Set an exit trap</i> int onexit(void (*func)(void));	LR-328
open	fcntl.h	<i>Open a level 1 file</i> int open(const char *name, int mode, int prot);	LR-330
opendir	dir.h	<i>Initiate a directory operation</i> DIR *opendir(const char *dirname);	LR-332

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
ovlyMgr		<i>Overlay manager call point</i> JMP ovlyMgr	LR-334
perror	stdio.h	<i>Print error message</i> void perror(const char *s);	LR-335
poserr	dos.h	<i>Print AmigaDOS error message</i> int poserr(const char *s);	LR-336
pow	math.h	<i>Raise a number to a power</i> double pow(double x, double y);	LR-339
pow2	math.h	<i>Raise 2 to a power</i> double pow2(double x);	LR-340
printf	stdio.h	<i>Formatted print to stdout</i> int printf(const char *fmt, ...);	LR-337
putc	stdio.h	<i>Put a character to a level 2 file</i> int putc(int fp, FILE *fp);	LR-341
putchar	stdio.h	<i>Put a character to stdout</i> int putchar(int c);	LR-342
putenv	stdlib.h	<i>Put a string into environment</i> int putenv(const char *env);	LR-343
putreg	dos.h	<i>Set up 68000-specific registers</i> void putreg(int reg, long x);	LR-345
puts	stdio.h	<i>Put a string to stdout</i> int puts(const char *s);	LR-347
qsort	stdlib.h	<i>Sort a data array</i> void qsort(void *a, size_t n, size_t size, int (*cmp) (const void *, const void *));	LR-348
raise	signal.h	<i>Generate a signal</i> int raise(int sig);	LR-349
rand	stdlib.h	<i>Generate a random number</i> int rand(void);	LR-350

Function Name	Header File	Description and Prototype	Page Ref.
rawcon	stdio.h	<i>Set or unset raw console input</i> <code>int rawcon(int flag);</code>	LR-352
rbrk	stdlib.h	<i>Release memory</i> <code>nt rbrk(void);</code>	LR-354
read	fcntl.h	<i>Read from a level 1 file</i> <code>int read(int fh, void *buffer, unsigned int length);</code>	LR-355
readdir	dir.h	<i>Read a directory element</i> <code>struct dirent *readdir(DIR *dfd);</code>	LR-356
readlocale	locale.h	<i>Read and initialize a new locale</i> <code>struct __locale *readlocale(const char *locale);</code>	LR-357
realloc	stdlib.h	<i>Reallocate memory</i> <code>void *realloc(void *b, size_t n);</code>	LR-358
remove	stdio.h	<i>Remove a file</i> <code>int remove(const char *name);</code>	LR-359
rename	stdio.h	<i>Rename a file</i> <code>int rename(const char *old, const char *new);</code>	LR-361
repmem	string.h	<i>Replicate values through a block</i> <code>void repmem(void *to, const void *vt, size_t nv, size_t nt);</code>	LR-363
rewind	stdio.h	<i>Reset a level 2 file position to its first byte</i> <code>void rewind(FILE *fp);</code>	LR-365
rewinddir	dir.h	<i>Reset to the start of the directory</i> <code>void rewinddir(DIR *dfd);</code>	LR-366
rlsmem	stdlib.h	<i>Release a memory block</i> <code>void rlsmem(void *p, unsigned int sbytes);</code>	LR-367
rlsml	stdlib.h	<i>Release a memory block</i> <code>void rlsml(void *p, long lbytes);</code>	LR-368
rmdir	dos.h	<i>Remove a directory</i> <code>int rmdir(const char *path);</code>	LR-369

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
rstmem	stdlib.h	<i>Reset the memory pool</i> void rstmem(void);	LR-370
sbrk	stdlib.h	<i>Allocate memory (unsigned)</i> void *sbrk(unsigned int sbytes);	LR-371
scanf	stdio.h	<i>Formatted input conversions</i> int scanf(const char *fmt, ...);	LR-372
seed48	math.h	<i>Set all 48 bits of an internal seed</i> unsigned short *seed48(unsigned short *seed);	LR-373
seekdir	dir.h	<i>Reposition a directory operation</i> void seekdir(DIR *dfd, long loc);	LR-374
setbuf	stdio.h	<i>Set the buffer mode for a level 2 file</i> void setbuf(FILE *fp, const char *buff);	LR-375
setjmp	setjmp.h	<i>Set long jump parameters</i> int setjmp(jmp_buf save);	LR-376
setlocale	locale.h	<i>Set locale information for a program</i> char *setlocale(int category, const char *locale);	LR-378
setmem	string.h	<i>Set a memory block to a value</i> void setmem(void *to, unsigned n, int n);	LR-380
setnbf	stdio.h	<i>Turn off I/O buffering for a level 2 file</i> void setnbf(FILE *fp);	LR-381
setvbuf	stdio.h	<i>Set the buffer mode for a level 2 file</i> int setvbuf(FILE *file, const char *buff, int type, size_t size);	LR-382
signal	signal.h	<i>Establish event traps</i> void (*signal)(int)(int newfun, void (*newfun)(int));	LR-384
sin	math.h	<i>Sine function</i> double sin(double x);	LR-386
sinh	math.h	<i>Hyperbolic sine function</i> double sinh(double x);	LR-387

Function Name	Header File	Description and Prototype	Page Ref.
sizmem	stdlib.h	<i>Get memory pool size</i> long sizmem(void);	LR-388
sprintf	stdio.h	<i>Formatted print to a string</i> int sprintf(char *s, const char *fmt, ...);	LR-389
sqsort	stdlib.h	<i>Sort an array of short integers</i> void sqsort(short *sa, size_t n);	LR-391
sqrt	math.h	<i>Square root function</i> double sqrt(double x);	LR-392
srand	stdlib.h	<i>Set the seed for the rand function</i> void srand(unsigned int seed);	LR-393
srand48	math.h	<i>Set high 32 bits of internal seed</i> void srand48(long hseed);	LR-395
sscanf	stdio.h	<i>Formatted input conversions</i> int sscanf(const char *ss, const char *fmt, ...);	LR-396
stacksize	dos.h	<i>Get the current stack size</i> size_t stacksize(void);	LR-397
stackused	dos.h	<i>Get the amount of stack in use</i> size_t stackused(void);	LR-398
stat	stat.h	<i>Get information on a file</i> int stat(const char *file, struct stat *st);	LR-399
stcarg	string.h	<i>Get an argument</i> int stcarg(const char *s, const char *b);	LR-401
stccpy	string.h	<i>Copy one string to another</i> int stccpy(char *to, const char *from, int n);	LR-403
std_i	string.h	<i>Convert a decimal string to an integer</i> int std_i(const char *in, int *ivalue);	LR-404
std_l	string.h	<i>Convert a decimal string to a long integer</i> int std_l(const char *in, long *lvalue);	LR-406

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
stcgfe	string.h	<i>Get the filename extension</i> int stcgfe(char *ext, const char *name);	LR-408
stcgfn	string.h	<i>Get a filename from a path</i> int stcgfn(char *node, const char *name);	LR-410
stcgfp	string.h	<i>Get the file path</i> int stcgfp(char *path, const char *name);	LR-411
stch_i	string.h	<i>Convert a hexadecimal string to an integer</i> int stch_i(const char *in, int *ivalue);	LR-413
stch_l	string.h	<i>Convert a hexadecimal string to a long integer</i> int stch_l(const char *in, long *lvalue);	LR-415
stci_d	string.h	<i>Convert an integer to a decimal string</i> int stci_d(char *out, int ivalue);	LR-417
stci_h	string.h	<i>Convert an integer to a hexadecimal string</i> int stci_h(char *out, int ivalue);	LR-419
stci_o	string.h	<i>Convert an integer to an octal string</i> int stci_o(char *out, int ivalue);	LR-421
stcis	string.h	<i>Count the number of string characters in the set</i> int stcis(const char *s, const char *b);	LR-423
stciscn	string.h	<i>Count the number of string characters not in the set</i> int stciscn(const char *s, const char *b);	LR-425
stcl_d	string.h	<i>Convert a long integer to a decimal string</i> int stcl_d(char *out, long lvalue);	LR-427
stcl_h	string.h	<i>Convert a long integer to a hexadecimal string</i> int stcl_h(char *out, long lvalue);	LR-429
stcl_o	string.h	<i>Convert a long integer to an octal string</i> int stcl_o(char *out, long lvalue);	LR-431
stclen	string.h	<i>Measure the length of a string</i> size_t stclen(const char *s);	LR-433
stco_i	string.h	<i>Convert an octal string to an integer</i> int stco_i(const char *in, int *ivalue);	LR-434

Function Name	Header File	Description and Prototype	Page Ref.
stco_l	string.h	<i>Convert an octal string to a long integer</i> int stco_l(const char *in, long *lvalue);	LR-436
stcpm	string.h	<i>Unanchored pattern match</i> int stcpm(const char *string, const char *pattern, char **match);	LR-438
stcpma	string.h	<i>Anchored pattern match</i> int stcpma(const char *string, const char *pattern);	LR-441
stcsma	string.h	<i>UNIX string pattern match (anchored)</i> int stcsma(const char *s, const char *p);	LR-444
stcu_d	string.h	<i>Convert an unsigned integer to a decimal string</i> int stcu_d(char *out, unsigned uvalue);	LR-445
stcul_d	string.h	<i>Convert an unsigned long to a decimal string</i> int stcul_d(char *out, unsigned long ulvalue);	LR-447
stpblk	string.h	<i>Skip blanks</i> char *stpblk(const char *p);	LR-449
stpbrk	string.h	<i>Find a break character in a string</i> char *stpbrk(const char *s, const char *b);	LR-450
stpchr	string.h	<i>Find a character in a string</i> char *stpchr(const char *s, int c);	LR-451
stpchrn	string.h	<i>Find a character not in a string</i> char *stpchrn(const char *s, int c);	LR-452
stpcpy	string.h	<i>Copy one string to another</i> char *stpcpy(char *to, const char *from);	LR-453
stpdate	string.h	<i>Convert a date array to a string</i> char *stpdate(char *p, int mode, const char *date);	LR-454
stpsym	string.h	<i>Get the next symbol from a string</i> char *stpsym(const char *s, char *sym, int symlen);	LR-456
stptime	string.h	<i>Convert a time array to a string</i> char *stptime(char *p, int mode, const char *time);	LR-458

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
stptok	string.h	<i>Get the next token from a string</i> char *stptok(const char *s, char *tok, int toklen, const char *brk);	LR-460
strbpl	string.h	<i>Build a string pointer list</i> int strbpl(char **s, int max, const char *t);	LR-462
streat	string.h	<i>Concatenate strings</i> char *strcat(char *to, const char *from);	LR-464
strchr	string.h	<i>Find a character in a string</i> char *strchr(const char *s, int c);	LR-465
strcmp	string.h	<i>Compare strings (case sensitive)</i> int strcmp(const char *a, const char *b);	LR-466
strcmpi	string.h	<i>Compare strings (case insensitive)</i> int strcmpi(const char *a, const char *b);	LR-469
strcoll	string.h	<i>Compare strings based on locale</i> int strcoll(const char *s1, const char *s2);	LR-470
strcpy	string.h	<i>Copy one string to another</i> char *strcpy(char *to, const char *from);	LR-471
strcspn	string.h	<i>Count the number of string characters not in the set</i> size_t strcspn(const char *s, const char *b);	LR-472
strdup	string.h	<i>Duplicate a string</i> char *strdup(const char *s);	LR-473
strerror	string.h	<i>Print the text for a given error number</i> char *strerror(int error);	LR-474
strftime	time.h	<i>Format a time string</i> size_t strftime(char *s, size_t maxsize, const char *format, const struct tm *timeptr);	LR-475
stricmp	string.h	<i>Compare strings (case insensitive)</i> int stricmp(const char *a, const char *b);	LR-478
strins	string.h	<i>Insert a string</i> void strins(char *to, const char *from);	LR-480

Function Name	Header File	Description and Prototype	Page Ref.
strlen	string.h	<i>Measure the length of a string</i> size_t strlen(const char *s);	LR-481
strlwr	string.h	<i>Convert a string to lowercase</i> char *strlwr(char *s);	LR-482
strmfe	string.h	<i>Make a filename with an extension</i> void strmfe(char *newname, const char *oldname, const char *ext);	LR-483
strmfn	string.h	<i>Make a filename from components</i> void strmfn(char *file, const char *drive, const char *path, const char *node, const char *ext);	LR-484
strmfp	string.h	<i>Make a filename from the path or node</i> void strmfp(char *name, const char *path, const char *node);	LR-486
strmid	string.h	<i>Return a substring from a string</i> int strmid(const char *source, char *dest, int pos, int len);	LR-487
strncat	string.h	<i>Concatenate strings (length limited)</i> char *strncat(char *to, const char *from, size_t n);	LR-488
strncmp	string.h	<i>Compare strings (length limited)</i> int strncmp(const char *a, const char *b, size_t n);	LR-490
strncpy	string.h	<i>Copy a string (length limited)</i> char *strncpy(char *to, const char *from, size_t n);	LR-492
strnicmp	string.h	<i>Compare strings (case insensitive, length limited)</i> int strnicmp(const char *a, const char *b, size_t n);	LR-493
strnset	string.h	<i>Set a string to a value (length limited)</i> char *strnset(char *s, int c, int n);	LR-495
strpbrk	string.h	<i>Find a break character in a string</i> char *strpbrk(const char *s, const char *b);	LR-496
strrchr	string.h	<i>Find the last occurrence of a character in a string</i> char *strrchr(const char *s, int c);	LR-497

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
strrev	string.h	<i>Reverse a character string</i> char *strrev(char *s);	LR-499
strset	string.h	<i>Set a string to a value</i> char *strset(char *s, int c);	LR-500
strsfm	string.h	<i>Split the filename</i> void strsfm(const char *file, char *drive, char *path, char *node, char *ext);	LR-501
strspn	string.h	<i>Count the number of string characters in the set</i> size_t strspn(const char *s, const char *b);	LR-503
strsrt	string.h	<i>Sort a string pointer list</i> void strsrt(char **s, int n);	LR-505
strstr	string.h	<i>Find a substring inside of a string</i> char *strstr(const char *s1, const char *s2);	LR-507
strtod	stdlib.h	<i>Convert a string to a double-precision floating-point number</i> double strtod(const char *nptr, char **endptr);	LR-509
strtok	string.h	<i>Get a token</i> char *strtok(char *s, const char *b);	LR-511
strtol	stdlib.h	<i>Convert a string to a long integer</i> long int strtol(const char *p, char **np, int base);	LR-513
strtoul	stdlib.h	<i>Convert a string to an unsigned long integer</i> unsigned long int strtoul(const char *p, char **np, int base);	LR-515
strupr	string.h	<i>Convert a string to uppercase</i> char *strupr(char *s);	LR-517
strxfrm	string.h	<i>Transform a string</i> size_t strxfrm(char *s1, const char *s2, size_t n);	LR-518
stpfp	string.h	<i>Parse a file path</i> int stpfp(const char *path, int *nx);	LR-520
_stub		<i>Default routine for undefined routines</i> void _stub(void);	LR-521

Function Name	Header File	Description and Prototype	Page Ref.
swmem	string.h	<i>Swap two memory blocks</i> void swmem(void *a, void *b, unsigned n);	LR-522
system	stdlib.h	<i>Call system command processor</i> int system(const char *cmd);	LR-523
tan	math.h	<i>Tangent function</i> double tan(double x);	LR-524
tanh	math.h	<i>Hyperbolic tangent function</i> double tanh(double x);	LR-525
tell	fcntl.h	<i>Get the level 1 file position</i> long tell (int fh);	LR-526
telldir	dir.h	<i>Get the directory position</i> long telldir(DIR *dfd);	LR-527
time	time.h	<i>Get the system time in seconds</i> time_t time(time_t *timeptr);	LR-528
__timecv	time.h	<i>Convert a time_t value to an AmigaDOS DateStamp</i> struct DateStamp *timecv(time_t t);	LR-529
timer	time.h	<i>Get the system clock with microseconds</i> int timer(unsigned int *clock);	LR-530
__tinymain	stdlib.h	<i>Special version of the __main function</i> void __stdargs __tinymain(char *line);	LR-531
tmpfile	stdio.h	<i>Open a temporary file stream</i> FILE *tmpfile(void);	LR-532
tmpnam	stdio.h	<i>Create a temporary filename</i> char *tmpnam(char *b);	LR-534
toascii	ctype.h	<i>Convert a character to ASCII</i> int toascii(int c);	LR-535
tolower	ctype.h	<i>Convert a character to lowercase</i> int tolower(int c);	LR-535

(continued)

Function Name	Header File	Description and Prototype	Page Ref.
toupper	ctype.h	<i>Convert a character to uppercase</i> int toupper(int c);	LR-535
tqsort	stdlib.h	<i>Sort an array of text pointers</i> void tqsort(char **ta, size_t n);	LR-537
__tzset	time.h	<i>Set the time zone variables</i> void __tzset(void);	LR-538
ungetc	stdio.h	<i>Push input character back</i> int ungetc(int c, FILE *fp);	LR-540
unlink	stdio.h	<i>Remove a file</i> int unlink (const char *name);	LR-542
utpack	time.h	<i>Pack UNIX time</i> long utpack(const char *x);	LR-544
utunpk	time.h	<i>Unpack UNIX time</i> void utunpk(long ut, char *x);	LR-546
va_arg	stdarg.h	<i>Get an argument from a varying-length argument list</i> (arg-type) va_arg(va_list ap, argtype);	LR-548
va_end	stdarg.h	<i>End varying-length argument list processing</i> void va_end(va_list ap);	LR-551
va_start	stdarg.h	<i>Begin varying-length argument list processing</i> void va_start(va_list ap, arg_name);	LR-552
vfprintf	stdarg.h, stdio.h	<i>Formatted write of a varying-length argument list to a file</i> int vfprintf(FILE *fp, const char *ctl, va_list args);	LR-553
vprintf	stdarg.h, stdio.h	<i>Formatted write of a varying-length argument list to stdout</i> int vprintf(const char *ctl, va_list args);	LR-555
vsprintf	stdarg.h, stdio.h	<i>Formatted write of a varying-length argument list to a string</i> int vsprintf(char *buf, const char *ctl, va_list args);	LR-557
wait	dos.h	<i>Wait for a single child process to complete</i> int wait(struct ProcID *procid);	LR-559

Function Name	Header File	Description and Prototype	Page Ref.
waitm	dos.h	<i>Wait for multiple child processes to complete</i> struct ProcID *waitm(struct ProcID **procid);	LR-559
wcstombs	stdlib.h	<i>Convert a wide-character string to a multibyte string</i> size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);	LR-560
wctomb	stdlib.h	<i>Map a wide character to multibyte character</i> int wctomb(char *s, wchar_t wc);	LR-561
write	fcntl.h	<i>Write to a level 1 file</i> int write(int fh, void *buffer, unsigned int length);	LR-562
_XCEXIT	stdlib.h	<i>Terminate the program</i> void _XCEXIT(long lcode);	LR-563



SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513

IntuitionBase=
GfxBase=Open



**This was brought to you
from the archives of**

<http://retro-commodore.eu>